UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/827,108	04/19/2004	Timothy Erickson Meehan	THER0009	1445

7590   03/25/2008

Michael C. King
LAW OFFICES OF RONALD M. ANDERSON
Suite 507
600 - 108th Avenue N.E.
Bellevue, WA 98004

EXAMINER
BROPHY, MATTHEW J

ART UNIT	PAPER NUMBER
2191	

MAIL DATE	DELIVERY MODE
03/25/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

PTOL-90A (Rev. 04/07)

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *19 April 2004*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-59* is/are pending in the application.

   4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-59* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☒ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *19 April 2004* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

   Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

   Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

   a)☐ All b)☐ Some * c)☐ None of:

   1.☐ Certified copies of the priority documents have been received.

   2.☐ Certified copies of the priority documents have been received in Application No. _____.

   3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

   * See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO/SB/08)
   Paper No(s)/Mail Date *8/27/2004*.

4) ☐ Interview Summary (PTO-413)
   Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

Claim Rejections - 35 USC § 112

DETAILED ACTION

1.   The following is a quotation of the second paragraph of 35 U.S.C. 112:

> The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

2.   Claims 20-25 and 28-34 rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

Regarding Claims 20-25, these claims refer to steps of "Claim 17". However, these steps are not included in Claim 17, but rather Claim 18. It is therefore suggested that Applicant amend these claims to correct this issue.

Regarding Claims 28-34, these claims refer to steps of "Claim 23" or "Claim 25". However, these steps are not included in Claim 23 or 25, but rather Claim 27. It is therefore suggested that Applicant amend these claims to correct this issue.

Claim Rejections - 35 USC § 102

3.   The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

4.   Claims 1-14, 39-53, 58 and 59, are rejected under 35 U.S.C. 102(b) as

being anticipated by Da Silva, Paulo Pinheiro. "Object Modelling of Interactive

Systems: The UMLi Approach." Thesis. University of Manchester. 2002. 217pp.

hereinafter Da Silva.

Regarding Claims 1 and 46, Da Silva teaches: A method (or system) for

using activity based notation to define interactions between a system and a user,

comprising the steps of: (a) separating the interaction between a user and a

system into a plurality of types of interactions, including: (i) inputter based

interactions that involve data provided by the user to the system **(e.g. Silva Page**

54, The Inputter category represents those components that can receive

information from users, such as text fields."); (ii) outputter based interactions

that involve data provided by the system to the user **(e.g. Silva Page 53, "The**

Displayer category represents those components that can present

information to users, such as labels."); (iii) invoker based interactions that

involve an action taken by the user to change a state of the system, and which

do not involve an exchange of data apparent to the user **(e.g. Silva Page 53,**

"The ActionInvoker sub-category of InteractionClass represents those

components that can receive information from users in the form of events,

such as buttons."); and (iv) selector based interactions that involve at least one

item of data being provided to the user by the system, and a subsequent

selection of at least one such item of data by the user **(e.g. Silva Page 54, "The**

Editor category represents those components that have the properties of

both Displayers and Inputters, such as combo boxes and selectable lists.");

(b) generating a statement for each interaction between a user and a system,

each statement containing elements providing information required to completely

describe the type of interaction and a nature of any information exchanged

between the user and the system as a consequence of the interaction, said

elements including at least: (i) a symbol indicating the type of interaction **(see**

e.g. Figure 4.4 Page 71); (ii) a textual description of the interaction **(e.g. Page**

71 John is looking for a book. He can check if such book is in the library

catalogue providing its title, authors, year, or a combination of this

information. Additionally, John can specify if he wants an exact or an

approximate match, and if the search should be over the entire catalogue

or the result of the previous query. Once the query has been submitted, the

system 4displays the details of the matching books, if any."); (iii) a definition

of a type of data exchanged between the user and the system **(e.g. Silva Page**

101 The type of the b variable in the value passing example is Bool (that

means, Boolean). Full LOTOS, or just LOTOS, provides a set of primitive

types for modelling simple data structures. For instance, the LOTOS

primitive nat denotes a type the domain of which must be a natural number.

LOTOS also provides the ability to specify more complex data structures

composed of primitive types and other complex data structures. For

example, the PERSON type definition presented as follows describes a type

that might be used by objects of a class PERSON"); and (iv) a definition of a

number of items of data exchanged during the interaction **(e.g. Silva Page 101**

The type of the b variable in the value passing example is Bool (that means,

Boolean). Full LOTOS, or just LOTOS, provides a set of primitive types for

modelling simple data structures. For instance, the LOTOS primitive nat

denotes a type the domain of which must be a natural number. LOTOS also

provides the ability to specify more complex data structures composed of

primitive types and other complex data structures. For example, the

PERSON type definition presented as follows describes a type that might

be used by objects of a class PERSON").

With regards to Claims 46, Da Silva further teaches:

(a) a computing device including: (i) an input device that receives input

from a user **(e.g. Page 61, "Indeed, many things happen when we are using**

an application: keys and buttons are pressed, the mouse is moved,

messages are sent to the network, etc."); (ii) a memory in which machine

instructions and data are stored **(inherent in the disclosure of Da Silva is that**

the UMLi would be stored on a computer memory); (iii) a display **(e.g. Page**

27, "Model-based user interface development environments (MB-UIDEs)

belong to a new category of tools for developing user interfaces (UIs)

through the construction of user interface models (UIMs). These models

describe, for example, the domain over which the user interface acts, the

tasks that the user interface supports, and various aspects of the display

presented to the user (e.g., the windows and interface components

used."); and (iv) a processor **(inherent in the use of computers in Da Silva's**

disclosure, e.g. In fact, computer-based tools may analyse big, non-complex models faster than small, complex models.")

Regarding Claims 2 and 47, Da Silva teaches: wherein the step of generating the statement comprises the step of including in the statement any filters defining restrictions upon the data exchanged between the user and the system **(e.g. Silva 115, "In addition to the internal sorts above, the ad type in Figure 5.10 specifies three other sorts to specify data items that interactors may exchange with users and other objects.")**.

Regarding Claims 3 and 48, Da Silva teaches: wherein the step of generating the statement comprises the step of including in the statement any conditions that must be met by the data exchanged between the user and the system, in accordance with predefined system rules **(e.g. Silva Page 80, "Figure 4.10 presents the UML Packages which are partially organised by both the participation of the metaclasses in the UML diagrams and the dependencies among the metaclasses. For each Package, the UML documentation provides three informal and complementary descriptions of the metamodel: the abstract syntax, well-formedness rules and modelling element semantics. The class diagrams, that can properly be called the UML metamodel, are part of the abstract syntax. The abstract syntax also provides a description in prose of each element that composes the UML metamodel. The well-formedness rules are written in OCL. These rules**

provide additional constraints concerning the metaclasses of the abstract
syntax").

Regarding Claims 4 and 49, Da Silva teaches: wherein the definition of the
number of items of data exchanged during the interaction indicates which items
of data are optionally exchanged and which items of data that are required to be
exchanged **(e.g. Da Silva Page 51, "Figure 3.6: The UML modelling of three
common interaction application behaviours. An order independent
behaviour is modelled in (a). A repeatable behaviour is modelled in (b). An
optional behaviour is modelled in (c).")**.

Regarding Claims 5 and 50, Da Silva teaches: wherein the step of
generating the statement comprises the step of generating the statement
according to rules that define relative positions of each element within the
statement **(e.g. Da Silva Page 80, "For each Package, the UML
documentation provides three informal and complementary descriptions of
the metamodel: the abstract syntax, well-formedness rules and modelling
element semantics."**.

Regarding Claims 6 and 51, Da Silva teaches: further comprising the step
of including the statements in a flow diagram **(e.g. Da Silva Page 76,
"Concerning the necessity of a complete decomposition of activities into
action states to achieve a description of object behaviours (UI Modelling
Difficulty 3), there are common functionalities related to interaction objects
that do not need to be modelled in detail to be understood. This fact is**

exploited by UMLi through the provision of five specialised stereotypes for object flows indicating the specification of such common functionalities. Object flows are informally called interaction object flows when they are related to interaction classes and have one of the stereotypes presented as follows.").

Regarding Claim 7, Da Silva teaches: wherein said flow diagram comprises at least one of an activity diagram and a flowchart **(e.g. Da Silva Page 78, "In the same way that the user interface diagram has simplified the appearance of UI presentations, the facilities of UMLi have also simplified the appearance of activity diagrams, as presented in Figure 4.8. These UMLi simplifications enable a discussion about the modelling of an activity diagram for the Search-Book functionality.").**

Regarding Claims 8 and 52, Da Silva teaches: further comprising the step of automatically generating a graphical user interface (GUI) form for guiding the user through each interaction with the system, said GUI form including at least one group for each statement **(e.g. Page 190, "The generation of user interface code from the user interface aspects modelled in UMLi models is the next natural candidate feature. Support for a collaborative design environment is another candidate feature that would have a significant impact for designers, especially for teams composed of application designers and user interface designers.").**

Regarding Claim 9 wherein the step of automatically generating the GUI form comprises the steps of: (a) mapping a GUI form to the flow diagram, such that each different statement in the flow diagram is separately mapped to a different group in the GUI form **(e.g. Da Silva Page 76, "Concerning the necessity of a complete decomposition of activities into action states to achieve a description of object behaviours (UI Modelling Difficulty 3), there are common functionalities related to interaction objects that do not need to be modelled in detail to be understood. This fact is exploited by UMLi through the provision of five specialised stereotypes for object flows indicating the specification of such common functionalities. Object flows are informally called interaction object flows when they are related to interaction classes and have one of the stereotypes presented as follows.")**; and (b) labeling each group in the GUI form based on a corresponding statement **(e.g. Da Silva Page "the object flow indicates that the state is responsible for an interaction between a user and the application. Thus, the ActionState can be an interaction where the user is invoking an object operation or visualising the result of an object operation. The ActionStates in the GetUserDetails activity in Figure 4.8 are examples of Inputters assigning values to some attributes of the UserQuery object from the domain. The 4 Feedback in Figure 4.8 is an example of a Displayer used for visualising the \Invalid Information" message, if required. As can be observed in Figure 4.8, two abstract operations specified in the APP (Figure 3.8) have been used along with**

these interaction objects. The setValue() operation is used by Displayers

and Editors for setting values to be presented to the users. The getValue()

operation is used by Inputters and Editors for passing values obtained

from users to domain objects. If the associated state is a PseudoState, the

object flow indicates the enactment of the interaction object for interaction.

If the interaction object is an instance of Container, such as the object

indicated by the UserDetailsCN in Figure 4.8, then its contained objects are

also activated for interaction.").

Regarding Claim 10, Da Silva teaches: further comprising one of the steps

of generating and modifying a graphical user interface (GUI) form, used for

guiding the user through each interaction with the system, as defined by the

generated statements **(e.g. Page 190, "The generation of user interface code**

from the user interface aspects modelled in UMLi models is the next

natural candidate feature. Support for a collaborative design environment

is another candidate feature that would have a significant impact for

designers, especially for teams composed of application designers and

user interface designers.").

Regarding Claim 11, Da Silva teaches:further comprising the step of

automatically modifying a flow diagram describing each interaction between the

user and the system, as the GUI form is modified **(e.g. Da Silva Page 76,**

"Concerning the necessity of a complete decomposition of activities into

action states to achieve a description of object behaviours (UI Modelling

Difficulty 3), there are common functionalities related to interaction objects

that do not need to be modelled in detail to be understood. This fact is

exploited by UMLi through the provision of five specialised stereotypes for

object flows indicating the specification of such common functionalities.

Object flows are informally called interaction object flows when they are

related to interaction classes and have one of the stereotypes presented as

follows.").

Regarding Claim 12, Da Silva teaches: further comprising the step of
automatically generating a flow diagram describing each interaction between the
user and the system, as the GUI form is generated **(e.g. Da Silva Page 76,**
"Concerning the necessity of a complete decomposition of activities into
action states to achieve a description of object behaviours (UI Modelling
Difficulty 3), there are common functionalities related to interaction objects
that do not need to be modelled in detail to be understood. This fact is
exploited by UMLi through the provision of five specialised stereotypes for
object flows indicating the specification of such common functionalities.
Object flows are informally called interaction object flows when they are
related to interaction classes and have one of the stereotypes presented as
follows.").

Regarding Claim 13, Da Silva teaches: wherein the step of automatically
generating a flow diagram comprises the steps of: (a) mapping the GUI form to
the flow diagram **(e.g. Da Silva Page 76, "Concerning the necessity of a**

complete decomposition of activities into action states to achieve a

description of object behaviours (UI Modelling Difficulty 3), there are

common functionalities related to interaction objects that do not need to be

modelled in detail to be understood. This fact is exploited by UMLi through

the provision of five specialised stereotypes for object flows indicating the

specification of such common functionalities. Object flows are informally

called interaction object flows when they are related to interaction classes

and have one of the stereotypes presented as follows."); (b) ensuring that

each interaction shown in the GUI form is present in the flow diagram (e.g. Da

Silva Page 76, "Concerning the necessity of a complete decomposition of

activities into action states to achieve a description of object behaviours

(UI Modelling Difficulty 3), there are common functionalities related to

interaction objects that do not need to be modelled in detail to be

understood. This fact is exploited by UMLi through the provision of five

specialised stereotypes for object flows indicating the specification of such

common functionalities. Object flows are informally called interaction

object flows when they are related to interaction classes and have one of

the stereotypes presented as follows."); and (c) for each interaction in the GUI

form, ensuring that information from statements corresponding to each activity in

the GUI form are included in a corresponding interaction in the flow diagram (e.g.

Da Silva Page 76, "Concerning the necessity of a complete decomposition

of activities into action states to achieve a description of object behaviours

(UI Modelling Difficulty 3), there are common functionalities related to

interaction objects that do not need to be modelled in detail to be

understood. This fact is exploited by UMLi through the provision of five

specialised stereotypes for object flows indicating the specification of such

common functionalities. Object flows are informally called interaction

object flows when they are related to interaction classes and have one of

the stereotypes presented as follows.").

Regarding Claim 14, Da Silva teaches: further comprising the step of

prompting a user to identify any statement information not automatically

recognized **(e.g. Da Silva Page 144, "To Do panel. This panel is located at**

the bottom left of Figure 6.1. Design critics provided by ArgoUML are

presented in this panel. This is a possible place for implementing some

constraints specified in the UMLi model. Indeed, rather than enforcing the

construction of consistent UML models from the beginning, ArgoUML

provides non-compulsory criticism facilities that may provide guidance for

building consistent models in an incremental way. The version of ArgoUML

that implements the UMLi extensions, v.0.8.1, does not make use of the

ArgoUML criticism facilities.").

Regarding Claim 39 and 58, Da Silva teaches: further comprising the step

of quantifying a number of action states in the flow diagram **(e.g. Da Silva Page**

77, "the object flow indicates that the state is responsible for an interaction

between a user and the application. Thus, the ActionState can be an

interaction where the user is invoking an object operation or visualising the

result of an object operation. The ActionStates in the GetUserDetails

activity in Figure 4.8 are examples of Inputters assigning values to some

attributes of the UserQuery object from the domain.").

Regarding Claims 40 and 59, Da Silva teaches: wherein the flow diagram

includes a plurality of blocks, at least some of which define at least one action

state, and wherein the step of quantifying a number of action states in the flow

diagram comprises the steps of: (a) parsing the flow diagram to identify each

block in the flow diagram that defines at least one action state **(e.g. Page 76,**

"An ☐interacts☐ interaction object flow relates an interaction object to an

ActionState or to a PseudoState3."); (b) for each block defining an action

state, determining if that block includes a statement corresponding to an

interaction between the user and the system, and if so, determining a number of

statements in that block **(e.g. Da Silva Page 77, "the object flow indicates that**

the state is responsible for an interaction between a user and the

application. Thus, the ActionState can be an interaction where the user is

invoking an object operation or visualising the result of an object

operation. The ActionStates in the GetUserDetails activity in Figure 4.8 are

examples of Inputters assigning values to some attributes of the

UserQuery object from the domain.").; (c) determining a number of blocks that

define at least one action state and do not include such a statement **(e.g. Page**

77, "If the associated state is a PseudoState, the object flow indicates the

enactment of the interaction object for interaction. If the interaction object

is an instance of Container, such as the object indicated by the

UserDetailsCN in Figure 4.8, then its contained objects are also activated

for interaction."); and (d) combining the number of blocks that define at least

one action state and do not include such a statement with the number of

statements in each block defining an action state that includes such a statement,

to quantity the number of action states in the flow diagram **e.g. Page 76, "An**

☐interacts☐ interaction object flow relates an interaction object to an

ActionState or to a PseudoState3.")

Regarding Claim 41, Da Silva teaches: further comprising the step of

determining a scope of the flow diagram **(e.g. Da Silva Page 125, "UML**

provides the ObjectFlowState and ClassifierInState constructs to specify

object flows, as discussed in Section 4.3.2. Thus, in the Connect Activity in

Figure 4.8, the uq ClassifierInState1 of type UserQuery is produced as a

result of the new UserQuery CreateAction ActionState. ObjectFlowStates

specify the incoming and outgoing of objects with respect to the scope of

an ActionState.").

Regarding Claim 42, Da Silva teaches: wherein the flow diagram

corresponds to at least one of a software based system and a hardware based

system **(Da Silva Page 95, "Further, such a semantics could be useful for**

implementing automated verification of models to identify incorrect uses of

the notation, as well automated as interpretation of models to generate

software code").

Regarding Claim 43, Da Silva teaches: wherein the step of determining a

scope of the diagram comprises the steps of quantifying a number of action

states in the flow diagram **(e.g. Da Silva Page 77, "the object flow indicates**

that the state is responsible for an interaction between a user and the

application. Thus, the ActionState can be an interaction where the user is

invoking an object operation or visualising the result of an object

operation. The ActionStates in the GetUserDetails activity in Figure 4.8 are

examples of Inputters assigning values to some attributes of the

UserQuery object from the domain."), and evaluating a level of effort

associated with the flow diagram **(e.g. Page 163, "Since decisions are**

specified in behavioural models, this is a metric for behavioural

complexity. Moreover, if the behavioural model is an activity diagram and A

is an activity in this activity diagram, then □(A) is the cyclomatic

complexity of A. The reasoning behind this metric is that □(A) corresponds

to the number of possible execution paths specified in an activity. The

assumption regarding this metric is that the higher the □(A), the more

complex the activity.") .

Regarding Claim 44, Da Silva teaches: wherein the flow diagram

comprises an activity diagram including a plurality of swimlanes, and wherein the

step of evaluating a level of effort associated with the flow diagram comprises the

steps of: (a) parsing the flow diagram to determine a number of paths contained

in the flow diagram **(e.g. Page 163, "Since decisions are specified in**

behavioural models, this is a metric for behavioural complexity. Moreover,

if the behavioural model is an activity diagram and A is an activity in this activity diagram, then □(A) is the cyclomatic complexity of A. The reasoning behind this metric is that □(A) corresponds to the number of possible execution paths specified in an activity. The assumption regarding this metric is that the higher the □(A), the more complex the activity."); (b) counting the plurality of swimlanes to determine a number of swimlanes in the flow diagram **(e.g. Page 163, "Since decisions are specified in behavioural models, this is a metric for behavioural complexity. Moreover, if the behavioural model is an activity diagram and A is an activity in this activity diagram, then □(A) is the cyclomatic complexity of A. The reasoning behind this metric is that □(A) corresponds to the number of possible execution paths specified in an activity. The assumption regarding this metric is that the higher the □(A), the more complex the activity.");** (c) identifying a number of crossings between the plurality of swimlanes **(e.g. Page 163, "Density of coupling between objects in diagrams (DCBOD). This is defined as the ratio of the level of CBO in the models and the total number of LOC of the PGML files representing the diagrams. The non-validated assumption is that high densities indicate that more relevant structural specification, e.g., structural complexity, can be represented by fewer graphical elements than with low densities.");** and (d) using the number of paths in the flow diagram, the number of swimlanes in the flow diagram, and the number of crossings between swimlanes to evaluate a level of effort associated with the flow diagram **(e.g. Page 163, "Since decisions are**

specified in behavioural models, this is a metric for behavioural

complexity. Moreover, if the behavioural model is an activity diagram and A

is an activity in this activity diagram, then □(A) is the cyclomatic

complexity of A. The reasoning behind this metric is that □(A) corresponds

to the number of possible execution paths specified in an activity. The

assumption regarding this metric is that the higher the □(A), the more

complex the activity.").

Regarding Claim 45, Da Silva teaches: wherein the flow diagram

comprises a flowchart, and wherein the step of evaluating a level of effort

associated with the flow diagram comprises the steps of: (a) parsing the flow

diagram to determine a number of paths contained in the flow diagram **(e.g.**

Page 163, "Since decisions are specified in behavioural models, this is a

metric for behavioural complexity. Moreover, if the behavioural model is an

activity diagram and A is an activity in this activity diagram, then □(A) is

the cyclomatic complexity of A. The reasoning behind this metric is that

□(A) corresponds to the number of possible execution paths specified in

an activity. The assumption regarding this metric is that the higher the

□(A), the more complex the activity."); and (b) using the number of paths in

the flow diagram to evaluate a level of effort associated with the flow diagram

(e.g. Page 163, "Since decisions are specified in behavioural models, this is

a metric for behavioural complexity. Moreover, if the behavioural model is

an activity diagram and A is an activity in this activity diagram, then □(A) is

the cyclomatic complexity of A. The reasoning behind this metric is that

☐(A) corresponds to the number of possible execution paths specified in

an activity. The assumption regarding this metric is that the higher the

☐(A), the more complex the activity.")

Regarding Claim 53, Da Silva teaches: wherein the machine instructions

further cause the processor: (a) enable a user to make changes to the GUI form

(e.g. Da Silva Page 143, "Editing panel. This panel is located at the top right

of Figure 6.1, and is where UMLi diagrams are constructed. This panel is

composed of the working area and the selection box. The selection box is

used for selecting a construct creator or an operator. Construct creators

are used for adding new components to the working area. Operators are

used for modifying constructs already created in the working area. The

contents of the selection box, in terms of construct creators and operators,

depends on the kind of diagram that is being edited. For instance, the

selection box in Figure 6.1 contains the construct creators for UI diagrams,

since this is the kind of diagram being edited."); and (b) in response to such

changes in the GUI form, automatically update the flow diagram to reflect such

changes **(e.g. Da Silva Page 143, "Editing panel. This panel is located at the**

top right of Figure 6.1, and is where UMLi diagrams are constructed. This

panel is composed of the working area and the selection box. The selection

box is used for selecting a construct creator or an operator. Construct

creators are used for adding new components to the working area.

Operators are used for modifying constructs already created in the working

area. The contents of the selection box, in terms of construct creators and

operators, depends on the kind of diagram that is being edited. For

instance, the selection box in Figure 6.1 contains the construct creators for

UI diagrams, since this is the kind of diagram being edited.").

Claim Rejections - 35 USC § 103

5.   The following is a quotation of 35 U.S.C. 103(a) which forms the basis for

all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described
> as set forth in section 102 of this title, if the differences between the subject matter sought to
> be patented and the prior art are such that the subject matter as a whole would have been
> obvious at the time the invention was made to a person having ordinary skill in the art to which
> said subject matter pertains. Patentability shall not be negatived by the manner in which the
> invention was made.

6.   Claims 15-18, 26, 27 and 54-56 are rejected under 35 U.S.C. 103(a) as

being unpatentable over Da Silva, Paulo Pinheiro. "Object Modelling of

Interactive Systems: The UMLi Approach." Thesis. University of Manchester.

2002. 217pp. hereinafter Da Silva in view of US Patent 6,854,089 Santee et al.

hereinafter Santee:

        Regarding Claims 15 and 54, Da Silva teaches the limitations of claim 6

and 51 as described above. However Da Silva does not teach: further comprising

the step of automatically generating test scripts based on the flow diagram.

However this limitation it taught by Santee **(Santee Column 2, Lines 43-53, "In**

another embodiment, the invention provides a system for testing

applications. The system includes an application mapper that

programmatically executes an application to generate a map of the

graphical user interface of the application. A script generator utilizes the

map to generate scripts that include instructions to test the application. An application tester executes the scripts to test the application. Typically, the application mapper generates the map by recursively performing actions on the graphical user interface of the application to identify new windows and add the new windows to the map."). In addition it would have been obvious to one of ordinary skill in the art at the time of the invention to combine the teachings of Da Silva with the script generation of Santee as both systems map the interaction with a graphical user interface and the system of Santee would allow test generation for a GUI modeled by Da Silva.

Regarding Claim 16, Santee further teaches: wherein the step of automatically generating test scripts based on the flow diagram comprises the steps of: (a) parsing the flow diagram such that each statement corresponding to a different interaction between the user and the system is identified **(Santee Column 4, Lines 40-54, "A script generator 107 utilizes application map 105 to generate scripts 109. In essence, script generator 107 utilizes the application map as a road map for scripts that exercise AUT 101. In a preferred embodiment, the script generator utilizes a genetic algorithm in order to generate the scripts. The genetic algorithm can include an iterative process of creating a population of scripts, running the population in conjunction with the AUT, ranking the population by fitness (e.g., such as complete code coverage), and then generating a new population. Such a script generator is described in full detail in U.S. patent application Ser. No. 08/655,149, filed May 30, 1996, which is hereby incorporated by reference.**

Other script generators that are known or have not yet been designed can

also be utilized to generate scripts from application map 105."); (b) providing

a graphical user interface (GUI) form used for guiding the user through each

interaction with the system and parsing diagram mapping information that maps

the flow diagram to the GUI form **(Santee Column 2, Lines 43-53, "In another**

embodiment, the invention provides a system for testing applications. The

system includes an application mapper that programmatically executes an

application to generate a map of the graphical user interface of the

application. A script generator utilizes the map to generate scripts that

include instructions to test the application. An application tester executes

the scripts to test the application. Typically, the application mapper

generates the map by recursively performing actions on the graphical user

interface of the application to identify new windows and add the new

windows to the map."); (c) parsing the GUI form to identify individual GUI

components **(Column 4, Lines 26-39, "An application map is a hierarchical**

representation of the graphical user interface. Application maps can

include numerous objects including windows, graphical user interface

objects, actions, transitions, and shortcuts (or pointers). Examples of

graphical user interface objects are buttons, sliders, check boxes, tab

controls, and the like. Additionally, examples of actions are left mouse

click, right mouse click, left mouse double click, key strokes, and the like.

Although the application map is most easily shown as it is displayed on the

monitor of a computer system an application map can be represented in

memory as a hierarchical representation of objects. Therefore, application

maps are not limited solely to the form in which they are displayed."); (d)

mapping each GUI component to a statement in the flow diagram **(Column 4,**

Lines 26-39, "An application map is a hierarchical representation of the

graphical user interface. Application maps can include numerous objects

including windows, graphical user interface objects, actions, transitions,

and shortcuts (or pointers). Examples of graphical user interface objects

are buttons, sliders, check boxes, tab controls, and the like. Additionally,

examples of actions are left mouse click, right mouse click, left mouse

double click, key strokes, and the like. Although the application map is

most easily shown as it is displayed on the monitor of a computer system

an application map can be represented in memory as a hierarchical

representation of objects. Therefore, application maps are not limited

solely to the form in which they are displayed."); (e) for each GUI component,

parsing the statement mapped to that GUI component **(Santee Column 4, Lines**

40-54, "A script generator 107 utilizes application map 105 to generate

scripts 109. In essence, script generator 107 utilizes the application map as

a road map for scripts that exercise AUT 101. In a preferred embodiment,

the script generator utilizes a genetic algorithm in order to generate the

scripts. The genetic algorithm can include an iterative process of creating a

population of scripts, running the population in conjunction with the AUT,

ranking the population by fitness (e.g., such as complete code coverage),

and then generating a new population. Such a script generator is described

in full detail in U.S. patent application Ser. No. 08/655,149, filed May 30,

1996, which is hereby incorporated by reference. Other script generators

that are known or have not yet been designed can also be utilized to

generate scripts from application map 105.");** and (f) generating a test script

for that GUI component **(Santee Column 4, Lines 40-54, "A script generator

107 utilizes application map 105 to generate scripts 109. In essence, script

generator 107 utilizes the application map as a road map for scripts that

exercise AUT 101. In a preferred embodiment, the script generator utilizes a

genetic algorithm in order to generate the scripts. The genetic algorithm

can include an iterative process of creating a population of scripts, running

the population in conjunction with the AUT, ranking the population by

fitness (e.g., such as complete code coverage), and then generating a new

population. Such a script generator is described in full detail in U.S. patent

application Ser. No. 08/655,149, filed May 30, 1996, which is hereby

incorporated by reference. Other script generators that are known or have

not yet been designed can also be utilized to generate scripts from

application map 105.").**

Regarding Claim 17, Santee further teaches: further comprising the step

of executing each test script to determine if the GUI form is displayed properly

**(e.g. Column 4, Lines 55-61, "Scripts 109 are designed to exercise AUT 101

and therefore include actions that are to be performed on the AUT. An

application tester 111 utilizes the scripts to exercise AUT 101 and can also

collect information regarding the execution of the AUT. For example,**

application tester 111 can collect statistics on code coverage, identify

bugs, find critical sections of code, and the like.").

Regarding Claim 26 and 56, Santee further teaches: further comprising

the step of performing a simulation of the flow diagram to enable the user to

determine if a GUI form for guiding the user through each interaction with the

system is correct **(e.g. Column 4, Lines 55-61, "Scripts 109 are designed to**

exercise AUT 101 and therefore include actions that are to be performed on

the AUT. An application tester 111 utilizes the scripts to exercise AUT 101

and can also collect information regarding the execution of the AUT. For

example, application tester 111 can collect statistics on code coverage,

identify bugs, find critical sections of code, and the like.").

Regarding Claims 18 and 27, Santee further teaches: wherein the step of

executing each test script to determine if the GUI form is displayed properly

comprises the steps of: (a) parsing the flow diagram to identify each statement

corresponding to a different interaction between the user and the system **(e.g.**

Column 5, Lines 21-35, "Actions 211 and 213 represent actions that can

occur (or have occurred) in the application due to user input. More

specifically, action 211 represents that a user launched the application and

action 213 represents that the user performed a left mouse click. Transition

215 represents that an event closed a window or dialog box. Other

transitions include an exit transition where an application is exited in a

normal fashion (e.g., clicking "exit") and crash transitions where the

application crashed during mapping. Still other behavior can be recorded;

like additional GUI behavior (e.g., size or position of a window) and states

of the application. Although it is not necessary to describe application map

201 in great detail, it should be evident that the application map is a

hierarchical mapping of the graphical user interface of the AUT."); (b)

mapping the GUI form to the flow diagram (e.g. Column 5, Lines 21-35,

"Actions 211 and 213 represent actions that can occur (or have occurred)

in the application due to user input. More specifically, action 211

represents that a user launched the application and action 213 represents

that the user performed a left mouse click. Transition 215 represents that

an event closed a window or dialog box. Other transitions include an exit

transition where an application is exited in a normal fashion (e.g., clicking

"exit") and crash transitions where the application crashed during

mapping. Still other behavior can be recorded; like additional GUI behavior

(e.g., size or position of a window) and states of the application. Although it

is not necessary to describe application map 201 in great detail, it should

be evident that the application map is a hierarchical mapping of the

graphical user interface of the AUT."); (c) retrieving and parsing each test

script corresponding to the flow diagram (e.g. Column 4, Lines 55-61, "Scripts

109 are designed to exercise AUT 101 and therefore include actions that

are to be performed on the AUT. An application tester 111 utilizes the

scripts to exercise AUT 101 and can also collect information regarding the

execution of the AUT. For example, application tester 111 can collect

statistics on code coverage, identify bugs, find critical sections of code,

and the like."); (d) displaying the GUI form **(Column 4, Lines 26-39, "An**

application map is a hierarchical representation of the graphical user

interface. Application maps can include numerous objects including

windows, graphical user interface objects, actions, transitions, and

shortcuts (or pointers). Examples of graphical user interface objects are

buttons, sliders, check boxes, tab controls, and the like. Additionally,

examples of actions are left mouse click, right mouse click, left mouse

double click, key strokes, and the like. Although the application map is

most easily shown as it is displayed on the monitor of a computer system

an application map can be represented in memory as a hierarchical

representation of objects. Therefore, application maps are not limited

solely to the form in which they are displayed."); (e) selecting a GUI

component from the GUI form **(Column 4, Lines 26-39, "An application map is**

a hierarchical representation of the graphical user interface. Application

maps can include numerous objects including windows, graphical user

interface objects, actions, transitions, and shortcuts (or pointers).

Examples of graphical user interface objects are buttons, sliders, check

boxes, tab controls, and the like. Additionally, examples of actions are left

mouse click, right mouse click, left mouse double click, key strokes, and

the like. Although the application map is most easily shown as it is

displayed on the monitor of a computer system an application map can be

represented in memory as a hierarchical representation of objects.

Therefore, application maps are not limited solely to the form in which they

are displayed."); (f) identifying a portion of the flow diagram corresponding to the GUI component selected **(e.g. Column 5, Lines 21-35, "Actions 211 and 213 represent actions that can occur (or have occurred) in the application due to user input. More specifically, action 211 represents that a user launched the application and action 213 represents that the user performed a left mouse click. Transition 215 represents that an event closed a window or dialog box. Other transitions include an exit transition where an application is exited in a normal fashion (e.g., clicking "exit") and crash transitions where the application crashed during mapping. Still other behavior can be recorded; like additional GUI behavior (e.g., size or position of a window) and states of the application. Although it is not necessary to describe application map 201 in great detail, it should be evident that the application map is a hierarchical mapping of the graphical user interface of the AUT.")**; (g) parsing each path in the portion of the flow diagram corresponding to the GUI component selected by the user, such that paths corresponding to interaction types are identified and parsed to identify corresponding statements and test scripts **(e.g. Column 4, Lines 55-61, "Scripts 109 are designed to exercise AUT 101 and therefore include actions that are to be performed on the AUT. An application tester 111 utilizes the scripts to exercise AUT 101 and can also collect information regarding the execution of the AUT. For example, application tester 111 can collect statistics on code coverage, identify bugs, find critical sections of code, and the like.")**; (h) identifying the interaction type and performing the

indicated interaction **(e.g. Column 4, Lines 55-61, "Scripts 109 are designed to exercise AUT 101 and therefore include actions that are to be performed on the AUT. An application tester 111 utilizes the scripts to exercise AUT 101 and can also collect information regarding the execution of the AUT. For example, application tester 111 can collect statistics on code coverage, identify bugs, find critical sections of code, and the like.")**; (i) executing the test script to determine if an error code results **(e.g. Column 4, Lines 55-61, "Scripts 109 are designed to exercise AUT 101 and therefore include actions that are to be performed on the AUT. An application tester 111 utilizes the scripts to exercise AUT 101 and can also collect information regarding the execution of the AUT. For example, application tester 111 can collect statistics on code coverage, identify bugs, find critical sections of code, and the like.")**; and (j) logging any resulting error code **(e.g. Column 4, Lines 55-61, "Scripts 109 are designed to exercise AUT 101 and therefore include actions that are to be performed on the AUT. An application tester 111 utilizes the scripts to exercise AUT 101 and can also collect information regarding the execution of the AUT. For example, application tester 111 can collect statistics on code coverage, identify bugs, find critical sections of code, and the like.")**.

Regarding Claim 55, Santee further teaches: wherein the machine instructions further cause the processor to execute each test script to identify any error codes that may be produced when the test script is executed **(e.g. Column 4, Lines 55-61, "Scripts 109 are designed to exercise AUT 101 and therefore**

include actions that are to be performed on the AUT. An application tester

111 utilizes the scripts to exercise AUT 101 and can also collect

information regarding the execution of the AUT. For example, application

tester 111 can collect statistics on code coverage, identify bugs, find

critical sections of code, and the like.").

7.   Claims 19-25 and 28-34 are rejected under 35 U.S.C. 103(a) as being

unpatentable over Da Silva, Paulo Pinheiro. "Object Modelling of Interactive

Systems: The UMLi Approach." Thesis. University of Manchester. 2002. 217pp.

hereinafter Da Silva in view of US Patent 6,854,089 Santee et al. hereinafter

Santee: as applied to claims 18 and 27 above, and further in view of US Patent

5,754,760 Warfield et al hereinafter Warfield.

        Regarding Claims 19 and 32, Da Silva in view of Santee teaches the

limitations of Claims 18 and 27 respectively above. These references do not

explicitly teach: wherein if a plurality of parameters can apply to affect the test

script, the test script is executed for each permutation and combination of

parameters that can apply to the test script. However, this limitation it taught by

Warfield **(Warfield Column 3, Lines 55-65, "The basic operation of the**

genetic algorithm 30 includes an iterative process of creating a population,

running the population in conjunction with the AUT 31, ranking the

population by fitness, and then generating a new population. The fitness

measures are used to indicate the performance of each script when

executed against the AUT 31. In particular, a fitness measure is a number

which indicates how close to achieving a defined goal (such as complete

code coverage) the script has come. It is important to be able to measure this closeness, or at least to be able to compare the relative closeness of two different scripts."). In addition it would have been obvious to one of ordinary skill in the art at the time of the invention to combine the teachings of Da Silva in view of Santee with the test generation techniques as Santee specifically references the Warfield patent in the quotations above, the two patents teach similar test script generation techniques and the Warfield reference teaches further implementation details of the test script generation originally seen in Santee.

Regarding Claims 20 and 33, Warfield further teaches: further comprising the step of determining if the GUI form displayed includes any GUI components for which a test script has not been executed, and if so, selecting that GUI component and implementing steps (f)-(j) of Claims 18 and 27 respectively **(Warfield Column 3, Lines 55-65, "The basic operation of the genetic algorithm 30 includes an iterative process of creating a population, running the population in conjunction with the AUT 31, ranking the population by fitness, and then generating a new population. The fitness measures are used to indicate the performance of each script when executed against the AUT 31. In particular, a fitness measure is a number which indicates how close to achieving a defined goal (such as complete code coverage) the script has come. It is important to be able to measure this closeness, or at least to be able to compare the relative closeness of two different scripts.")**.

Regarding Claims 21 and 34, Warfield further teaches: further comprising

the step of determining if an additional GUI form is being displayed, and if so,

selecting a GUI component from the additional GUI form, and implementing

steps (f)-(j) of Claims 18 and 27 respectively for the GUI component selected

from the additional GUI form **(Warfield Column 3, Lines 55-65, "The basic**

operation of the genetic algorithm 30 includes an iterative process of

creating a population, running the population in conjunction with the AUT

31, ranking the population by fitness, and then generating a new

population. The fitness measures are used to indicate the performance of

each script when executed against the AUT 31. In particular, a fitness

measure is a number which indicates how close to achieving a defined goal

(such as complete code coverage) the script has come. It is important to be

able to measure this closeness, or at least to be able to compare the

relative closeness of two different scripts.").

Regarding Claims 22 and 28, Warfield further teaches: wherein if the

interaction type identified in step (h) of Claims 18 and 27 respectively is an

inputter based interaction, further comprising the step of inputting random data

before executing the script **(e.g. Column 4, Lines 5-20, "The process of**

running a population entails executing the AUT 31 in response to each test

script in the population. In particular, each test script is used to drive the UI

or an API of the AUT 31. In the preferred embodiment, the fitness measure

is a code coverage value generated by code coverage analyzer 32.

However, as will be described below, other types of fitness measures are

possible. Code coverage analyzer 32 monitors the output of AUT 31 and

determines the amount of code of AUT 31 that was executed in response to

each test script. Code coverage analyzer 32 then provides to the genetic

algorithm 30 a value (i.e., a fitness value) indicating the amount of code

coverage. The genetic algorithm 30 generates the test scripts of each

population based upon the previous population, including its fitness

values, except in the case of the first population, which is generated

randomly.").

Regarding Claims 23 and 29, Warfield further teaches: wherein if the

interaction type identified in step (h) of Claims 18 and 27 respectively is an

outputter based interaction, further comprising the steps of parsing the output,

and applying any filters and conditions before executing the script **(e.g. Column**

5, Lines 5-14, "As mentioned above, each test script comprises one or

more test cases. Each test case is an independently verifiable operation.

An example of a test case is the act of printing. A test case for printing

must take the UI through all of the steps which are necessary to obtain

output from a printer. Other types of test cases will test: menu commands;

mouse gestures, such as selection or drawing; editing interactions, such

as entering a name in a field; standard window interactions, such as

resizing or scrolling a window; or OLE (Object Linking and Embedding)

interactions.").

Regarding Claims 24 and 30, Warfield further teaches: wherein if the

interaction type identified in step (h) of Claims 18 and 27 respectively is an

invoker based interaction, further comprising the steps of invoking the interaction, and applying any filters and conditions before executing the script **(e.g. Column 5, Line 64-Column 6, Line 6, "For mouse gestures, a state machine definition will capture the basic states and transitions required to perform a selection gesture in the AUT 31. These gestures might include gestures which select a tool and draw. For menu commands, a state machine definition will model the complete process of menu selection, filling in a dialog box, and command completion. For entering a name in a field, a state machine definition will include the possibility of editing keystrokes, such as backspace, delete, home or end, as well as pasting from a clipboard, etc.")**.

Regarding Claims 25 and 31, Warfield further teaches: wherein if the interaction type identified in step (h) of Claims 18 and 27 respectively is a selector based interaction, further comprising the steps of generating all possible selection sets, and applying any filters and conditions to each selection set before executing the script for each selection set **(Warfield Column 3, Lines 55-65, "The basic operation of the genetic algorithm 30 includes an iterative process of creating a population, running the population in conjunction with the AUT 31, ranking the population by fitness, and then generating a new population. The fitness measures are used to indicate the performance of each script when executed against the AUT 31. In particular, a fitness measure is a number which indicates how close to achieving a defined goal (such as complete code coverage) the script has come. It is important to be**

able to measure this closeness, or at least to be able to compare the

relative closeness of two different scripts.").

8.   Claims 35 and 57 rejected under 35 U.S.C. 103(a) as being unpatentable

over Da Silva, Paulo Pinheiro. "Object Modelling of Interactive Systems: The

UMLi Approach." Thesis. University of Manchester. 2002. 217pp. hereinafter Da

Silva in view of US PG Publication 2005/0210397 Kanai et al. hereinafter Kanai.

9.   Regarding Claims 35 and 57, Da Silva teaches the limitations of Claim 6

and 51 as described above. Da Silva does not teach: further comprising the step

of producing graphical user interface (GUI) hardware components from a

computer aided design (CAD) drawing, the GUI hardware components being

configured to guide the user through each interaction with the system. However,

this limitation is taught by Kanai. **(e.g. Kanai Paragraph [0056] "A housing**

mock-up of the VWC system is prepared according to sequences shown in

FIGS. 12 and 13. In step (a) of FIG. 12, a VWC housing is designed by CAD.

In step (b), a housing mock-up without operation buttons is made by laser

lithography. The operation buttons are attached to the housing mock-up

later. In step (c), the operation buttons (including push buttons, seesaw

buttons, and cross buttons) are made. These buttons are attached to

optional locations on the housing mock-up. FIG. 13 shows the details of

making operation buttons. In step (a) of FIG. 13, each operation button is

designed by CAD. In step (b), a hole to embed an RFID chip is designed by

CAD on the operation button."). In addition it would have been obvious to one

of ordinary skill in the art at the time of the invention to combine the teachings of

Da Silva with the CAD UI design features of Kanai as both system teach the

modeling of user interfaces and the modeling techniques of could be applied to

hardware systems by using the CAD system of Kanai.

10.   Claims 36-38 are rejected under 35 U.S.C. 103(a) as being unpatentable

over Da Silva, Paulo Pinheiro. "Object Modelling of Interactive Systems: The

UMLi Approach." Thesis. University of Manchester. 2002. 217pp. hereinafter Da

Silva in view of US PG Publication 2005/0210397 Kanai et al. hereinafter Kanai

and further in view of US Patent 5,005,119 Rumbaugh et al hereinafter

Rumbaugh.

Regarding Claim 36, Da Silva in view of Kanai teach the limitations of Claim 35

as described above. In addition Kanai teaches: wherein the step of producing

GUI hardware components comprises the steps of: and (c) using the CAD

drawing to control equipment to produce the hardware components **Kanai**

Paragraph [0056] "A housing mock-up of the VWC system is prepared

according to sequences shown in FIGS. 12 and 13. In step (a) of FIG. 12, a

VWC housing is designed by CAD. In step (b), a housing mock-up without

operation buttons is made by laser lithography. The operation buttons are

attached to the housing mock-up later. In step (c), the operation buttons

(including push buttons, seesaw buttons, and cross buttons) are made.

These buttons are attached to optional locations on the housing mock-up.

FIG. 13 shows the details of making operation buttons. In step (a) of FIG.

13, each operation button is designed by CAD. In step (b), a hole to embed

an RFID chip is designed by CAD on the operation button."). However,

neither reference teaches: (a) mapping a CAD drawing to the flow diagram, such

that each different statement in the flow diagram is separately mapped to a

different group in the CAD drawing (b) labeling each group in the CAD drawing

based on a corresponding statement. However these limitations are taught by

Rumbuagh. (a) mapping a CAD drawing to the flow diagram, such that each

different statement in the flow diagram is separately mapped to a different group

in the CAD drawing **(e.g. Column 3, Lines 10-19, "In a preferred embodiment**

of the present invention, the user interfaces with the control system

through a graphical display device, such as a high resolution cathode ray

tube. The control system provides a visual representation of the program

sequence and program input/output data set requirements in the form of a

flowgraph. The control system of the present invention implemented with

the flowgraph representation is referred to hereinafter as the flowgraph

system. The engineer is apprised by images on the flowgraph of the status

of a current design, i.e. to what point the design task has progressed in the

program sequence. Further, the engineer can interact with the flowgraph

graphical display, e.g., by means of a cursor positioning device such as a

mouse, to initiate program execution or create new design versions.") ; (b)

labeling each group in the CAD drawing based on a corresponding statement

(e.g. Column 3, Lines 10-19, "In a preferred embodiment of the present

invention, the user interfaces with the control system through a graphical

display device, such as a high resolution cathode ray tube. The control

system provides a visual representation of the program sequence and

program input/output data set requirements in the form of a flowgraph. The control system of the present invention implemented with the flowgraph representation is referred to hereinafter as the flowgraph system. The engineer is apprised by images on the flowgraph of the status of a current design, i.e. to what point the design task has progressed in the program sequence. Further, the engineer can interact with the flowgraph graphical display, e.g., by means of a cursor positioning device such as a mouse, to initiate program execution or create new design versions."). In addition it would have been obvious to one of ordinary skill in the art to at the time of the invention to combine these references as Da Silva teaches a method of modeling User interfaces including interaction flows, Kanai teaches a method of designing UIs based on CAD drawings, and Rumbaugh teaches a method of modeling interaction flows with CAD drawings. The system of Rumbaugh would allow the application of the interaction flow modeling shown in Da Silva with the CAD-UI design of Kanai, and the combination would have been obvious as such a combination would allow the UI models of Da Silva to be implemented in a hardware interface, such as the cellular telephone of Kanai.

Regarding Claims 37, Kanai further teaches: further comprising the steps of: (a) providing a computer aided design (CAD) drawing of a graphical user interface (GUI) hardware component configured to guide the user through each interaction with the system **Kanai Paragraph [0056] "A housing mock-up of the VWC system is prepared according to sequences shown in FIGS. 12 and 13. In step (a) of FIG. 12, a VWC housing is designed by CAD. In step**

(b), a housing mock-up without operation buttons is made by laser

lithography. The operation buttons are attached to the housing mock-up

later. In step (c), the operation buttons (including push buttons, seesaw

buttons, and cross buttons) are made. These buttons are attached to

optional locations on the housing mock-up. FIG. 13 shows the details of

making operation buttons. In step (a) of FIG. 13, each operation button is

designed by CAD. In step (b), a hole to embed an RFID chip is designed by

CAD on the operation button."). However, Kanai and Da Silva do not teach:

and (b) automatically generating a flow diagram describing each interaction

between the user and the system based on the GUI hardware component.

However this limitation is taught by Rumbaugh **(e.g. Column 3, Lines 10-19, "In**

a preferred embodiment of the present invention, the user interfaces with

the control system through a graphical display device, such as a high

resolution cathode ray tube. The control system provides a visual

representation of the program sequence and program input/output data set

requirements in the form of a flowgraph. The control system of the present

invention implemented with the flowgraph representation is referred to

hereinafter as the flowgraph system. The engineer is apprised by images

on the flowgraph of the status of a current design, i.e. to what point the

design task has progressed in the program sequence. Further, the engineer

can interact with the flowgraph graphical display, e.g., by means of a

cursor positioning device such as a mouse, to initiate program execution

or create new design versions.").

Regarding Claim 38, Rumbaugh further teaches: wherein the step of

automatically generating a flow diagram comprises the steps of: (a) mapping the

CAD drawing to a flow diagram **(e.g. Column 3, Lines 10-19, "In a preferred**

embodiment of the present invention, the user interfaces with the control

system through a graphical display device, such as a high resolution

cathode ray tube. The control system provides a visual representation of

the program sequence and program input/output data set requirements in

the form of a flowgraph. The control system of the present invention

implemented with the flowgraph representation is referred to hereinafter as

the flowgraph system. The engineer is apprised by images on the

flowgraph of the status of a current design, i.e. to what point the design

task has progressed in the program sequence. Further, the engineer can

interact with the flowgraph graphical display, e.g., by means of a cursor

positioning device such as a mouse, to initiate program execution or create

new design versions."); (b) ensuring that each interaction shown in the CAD

drawing is present in the flow diagram **(e.g. Column 3, Lines 10-19, "In a**

preferred embodiment of the present invention, the user interfaces with the

control system through a graphical display device, such as a high

resolution cathode ray tube. The control system provides a visual

representation of the program sequence and program input/output data set

requirements in the form of a flowgraph. The control system of the present

invention implemented with the flowgraph representation is referred to

hereinafter as the flowgraph system. The engineer is apprised by images

on the flowgraph of the status of a current design, i.e. to what point the

design task has progressed in the program sequence. Further, the engineer

can interact with the flowgraph graphical display, e.g., by means of a

cursor positioning device such as a mouse, to initiate program execution

or create new design versions."); and (c) for each interaction in the CAD

drawing, ensuring that information from statements corresponding to each

interaction in the CAD drawing are included in the corresponding interaction in

the flow diagram **(e.g. Column 3, Lines 10-19, "In a preferred embodiment of**

the present invention, the user interfaces with the control system through a

graphical display device, such as a high resolution cathode ray tube. The

control system provides a visual representation of the program sequence

and program input/output data set requirements in the form of a flowgraph.

The control system of the present invention implemented with the

flowgraph representation is referred to hereinafter as the flowgraph

system. The engineer is apprised by images on the flowgraph of the status

of a current design, i.e. to what point the design task has progressed in the

program sequence. Further, the engineer can interact with the flowgraph

graphical display, e.g., by means of a cursor positioning device such as a

mouse, to initiate program execution or create new design versions.").

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to MATTHEW J. BROPHY whose telephone number is 571-270-1642. The examiner can normally be reached on Monday-Thursday 8:00AM-5:00 PM EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Wei Zhen can be reached on (571) 272-3708. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

MJB

3/11/2008
/Wei Zhen/

Supervisory Patent Examiner, Art Unit 2191